PROYECTO FIN DE CARRERA

Ingeniería Industrial

Programa para la resolución de sistemas de ecuaciones no lineales

Universidad de Zaragoza **Centro Politécnico Superior** **Dpto. Matemática aplicada**

Director de Proyecto: **Dr. D. Francisco José Gaspar Lorenz**

Ponente: **Dr. D. María Cruz Lopez de Silanes Busto**

Autor: **D. Pablo Salinas Cortés**

Diciembre 2010

Área de Matemática aplicada

Departamento de Matemática aplicada

Programa para la resolución de sistemas de ecuaciones no lineales

Resumen

Viendo los problemas que presentaba la versión del Engineering Equation Solver, que la Universidad dejaba al alumnado, tales como no permitir un punto inicial, fecha de caducidad de la aplicación, que solamente está en inglés y que únicamente funciona en Windows; nos decidimos a crear una plataforma que intentase, en la medida de lo posible, solventar estos errores y sobre todo dar una alternativa.

Comenzamos este proyecto tras estudiar los lenguajes de programación que podían ser más apropiados para nuestro propósito y documentarnos sobre los métodos para resolver sistemas de ecuaciones. Pero considerando la complejidad ante la que nos encontrábamos, resolvimos que la mejor solución sería realizar el programa en *software* libre[17], añadiendo la posibilidad de que en un futuro, más gente pueda ayudar a mejorar este programa, o incluso sacar sus propias versiones, aportando un valor añadido a nuestro proyecto, puesto que no es una caja negra.

El programa realizado tiene implementados métodos para dividir sistemas de ecuaciones en subsistemas de ecuaciones, permitiendo así resolver grandes sistemas de una manera más rápida y eficiente. Además, incluye cuatro métodos de optimización globalmente convergentes, lo que significa que rara vez deberemos escoger un punto inicial para poder obtener la solución de su problema. Sin embargo, hemos incluido varios menús para que se puedan modificar los parámetros suficientes, como el punto inicial del algoritmo o el radio de región de confianza, para que estos algoritmos se adapten a distintos tipos de problemas no lineales.

Se ha diseñado una completa interfaz de usuario, con capacidad para deshacer, rehacer, cortar, copiar, pegar, buscar, exportar a pdf, imprimir, guardar y abrir documentos del tipo del programa.
Hemos creado e incluido dentro del programa una ayuda completa para que el usuario sea capaz de utilizar el programa y entienda su valor intrínseco.

Finalmente hemos incluido una base de datos con propiedades termodinámicas de distintas sustancias, que puede ser ampliada por el usuario.

Memoria

Índice

1. Introducción

En Ingeniería, podemos encontrar sistemas de ecuaciones no lineales tanto en electricidad, como en mecánica, pasando por otras ramas como las ondas electromagnéticas y la tecnología nuclear. Es muy importante, por tanto, tener las herramientas apropiadas para enfrentarse a estos problemas, nada sencillos de resolver. De hecho, la gran mayoría de ellos solamente pueden solucionarse mediante herramientas basadas en cálculo numérico.

Este proyecto se inscribe en este aspecto, con el objetivo de crear una plataforma con la que poder trabajar fácilmente con sistemas de ecuaciones. Para ello, hemos creado una plataforma de *software* que incluya su propia interfaz de usuario y que no dependa de otros programas. De esta manera, únicamente depende de la máquina virtual de Java*, lo cual es un requisito casi imprescindible trabajando con este lenguaje de programación. El resto de dependencias quedan incluidas dentro del propio programa. En consecuencia, obtenemos un *software* compacto e independiente.

Para darle mayor utilidad en el mundo ingenieril, hemos incluido la capacidad de trabajar con una base de datos con propiedades termodinámicas, de tal manera que el usuario pueda ampliarla y modificarla. Como vamos a ver a lo largo de estas líneas, la figura del usuario va a ser muy importante en el desarrollo del proyecto ya que queremos presentar al final un programa en *software* libre, es decir, abierto a que terceros puedan modificarlo, ampliarlo, corregirlo o, incluso, elaborar sus propias versiones independientes del original. Esta es la principal razón por la que incluimos la documentación interna del programa en inglés, pues es el idioma más utilizado en este ámbito y así podremos llegar a un público más amplio. En esa misma línea de facilitar el acceso al usuario y a posibles programadores, en la ayuda del programa hemos añadido una pequeña explicación de cómo funciona el programa a nivel interno a la hora de resolver los sistemas de ecuaciones, además de facilitar la ayuda básica sobre la manera de usar el programa en sí.

Ante las diferentes necesidades que han surgido a lo largo del proyecto, hemos acudido también al *software* libre, ya que hemos utilizado diferentes librerías para las diversas partes del proyecto, con el consiguiente ahorro de tiempo que conlleva no tener que crearlas. Quedan reflejadas en el apartado 3, Información sobre el programa.

La elección del lenguaje de programación vino secundada por los objetivos del proyecto, pues Java era el que mejor y más fácilmente permitía cumplir los objetivos, como veremos posteriormente.

* Definición de máquina virtual en el punto 2.4.

Este proyecto pretende ser una alternativa al programa de resolución de ecuaciones comercial Engineering Equation Solver, del cual hemos sacado algunas ideas para llevar a cabo este proyecto.

1.1. Métodos

Para la resolución de los sistemas, hemos visto la necesidad de documentarnos sobre la resolución numérica de ecuaciones[1,2,3,4,5,6,15] con computadores[16]; y los métodos existentes para concluir que los métodos que mejor se adaptarían a nuestras necesidades serían los siguientes:

– Métodos de región de confianza: son algoritmos de optimización con convergencia global. Hemos utilizado la implementación escrita en Java Nonlínear Optimization Java Package, que incluye también un modo de resolución por búsqueda lineal en vez de región de confianza. Ambos modos de resolución resultan interesantes en nuestro proyecto.

– Algoritmo de Tarjan: es un método que busca componentes fuertemente conexas en grafos.

– LU/QR: son métodos no numéricos para la resolución de sistemas de ecuaciones lineales. Se utilizan en cada iteración de los métodos de región de confianza. Hemos empleado las implementaciones de Nonlinear Optimization Java Package y CommonsMath.

1.2. Contenido de los documentos

– Introducción teórica: recopilamos las bases matemáticas sobre las que se basa este proyecto.

– Información sobre el programa: explicamos las causas que han motivado la elección del lenguaje de programación frente a los demás, las dependencias y cómo está distribuido el programa.

– Funcionamiento del programa: analizamos el proceso de resolución de las ecuaciones y de otras operaciones.

– Menús del programa: mostramos información sobre los distintos menús de los que dispone el programa para configurarlo, trabajar con la base de datos o la ayuda.

– Otras funciones: explicamos cómo realizamos operaciones menores como imprimir o exportar a PDF.

– Conclusiones.

2. Introducción teórica

2.1. Ecuaciones y sistemas de ecuaciones

Como este proyecto se ubica en el ámbito de las ecuaciones, la primera tarea va a ser definir el término en sí, concluyendo que una ecuación es la representación matemática mediante símbolos de dos expresiones algebraicas en las que aparecen dos términos iguales[15].

Por otra parte, una función expresa la dependencia entre dos cantidades[15]. Sin embargo resulta más interesante el hecho de que una función valga cero para los valores que resuelven su ecuación equivalente.

Por ejemplo: Ecuación: $x+y=5$ **(1)** → Función equivalente: $f(x,y)=x+y-5$ **(2)**

Dentro de las ecuaciones podemos diferenciar entre ecuaciones lineales y ecuaciones no lineales:

- Lineales: son ecuaciones algebraicas donde cada término es o bien una constante o bien el producto de una constante por una variable elevada a uno. Estas ecuaciones se llaman lineales debido a que representadas en ejes cartesianos son líneas rectas[15].

Por ejemplo: $4+3*x=7$ **(3)**

- No lineales: todas las ecuaciones que no se correspondan con lineales[15]. Un ejemplo sería: $x^2+x+4=\exp(x)$ **(4)**

Independientemente de si son lineales o no lineales, las ecuaciones pueden agruparse formando sistemas de ecuaciones donde unas ecuaciones dependan de las demás. Para resolverlas, deben encontrarse los valores exactos de las variables que cumplan todas las ecuaciones.

Definiciones importantes sobre las funciones:

- Continua: una función es continua si se dan pequeñas variaciones de valor para puntos cercanos en el dominio. Coloquialmente, las reconocemos como aquellas que pueden dibujarse sin levantar el lápiz del papel[15].
- Diferenciable: una función es diferenciable si se puede derivar en cualquier dirección y puede aproximarse al menos hasta el primer orden por una función afín[15].
- Jacobiana: es una matriz que contiene las derivadas parciales de las funciones del sistemas de ecuaciones[15].

Las ecuaciones y los sistemas de ecuaciones pueden resolverse analíticamente o numéricamente:

- Analíticamente: se saca el valor exacto de la ecuación. La resolución de sistemas se restringen a sistemas de ecuaciones lineales[15].
- Numéricamente: se realiza mediante métodos iterativos en los que poco a poco se va

aproximando a la solución. Necesitan datos adicionales como un punto inicial del que partir o incluso un intervalo donde la solución existe. Estos métodos sirven tanto para lineales como no lineales[15].

2.2. Métodos numéricos de resolución de sistemas de ecuaciones no lineales

En primer lugar, es necesario entender la dificultad que requiere la elección de un algoritmo u otro. Hay que explicar la diferencia e importancia del tipo de convergencia que pueden tener estos métodos:

– Convergencia local: el algoritmo solamente alcanzará la solución del problema si el punto inicial está lo suficientemente cerca de la solución. Pero esto depende del tipo de problema al que nos enfrentemos, pues lo que es suficientemente cerca para unos no lo es para otros[2].

– Convergencia global: será aquel que sea capaz de alcanzar la solución, independientemente del punto inicial de algoritmo. Desgraciadamente esto no es posible debido a que las funciones no lineales pueden, por ejemplo, no existir en ciertas zonas y empezar ahí el algoritmo impide por tanto alcanzar la solución. En consecuencia, se dice que un método es globalmente convergente si es capaz de comenzar desde un punto muy lejano a la solución y alcanzarla. Nuevamente, no se puede especificar qué es lejano, pues depende del tipo de problema al que nos enfrentemos[2].

Como vemos, es muy importante que el algoritmo sea globalmente convergente ya que aunque no nos garantice totalmente la posibilidad de alcanzar una solución, sí que nos va a dar más garantías de que la encontrará. Además nos interesa que el algoritmo sirva para funciones sin límites, es decir, que no le tengamos que decir al algoritmo que la solución está en un intervalo determinado, ya que, se produciría un aumento del trabajo del usuario debido a que tendría que suponer intervalos donde va a estar la solución de todas las variables.

Daremos por hecho que las funciones con las que vamos a trabajar son continuas y derivables, así como que la Jacobiana de las mismas, no es singular en ningún punto. Si alguna de estas premisas no se diera no se podría garantizar que el algoritmo funcione.

En este proyecto vamos a usar los métodos globalmente convergentes de optimización conocidos como Región de confianza y de búsqueda lineal*.

2.3. Grafos

Un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de dicho conjunto[15].

* Para más información sobre estos métodos y su modo de implementación mirar el anexo 2.

En un grafo, los nodos o vértices se representan con puntos y con líneas, las uniones entre ellos, aristas.

Los grafos permiten estudiar las relaciones entre unidades que interactúan entre ellas, por ejemplo, se pueden representar con grafos unas líneas de suministro, una red de ordenadores o eléctrica, etc. En el caso de las líneas de suministro, los nodos representarían las ciudades y las líneas las uniones entre ellas.

2.3.1. Tipos de grafos[7,15]

Hay 4 tipos de grafos:

- Grafo no dirigido: es un conjunto de pares no ordenados.
- Grafo dirigido: es un conjunto de pares ordenados.
- Pseudografo: un grafo no dirigido que acepta bucles.
- Pseudografo dirigido: un grafo dirigido que acepta bucles.

2.3.2. Propiedades de los grafos[15]

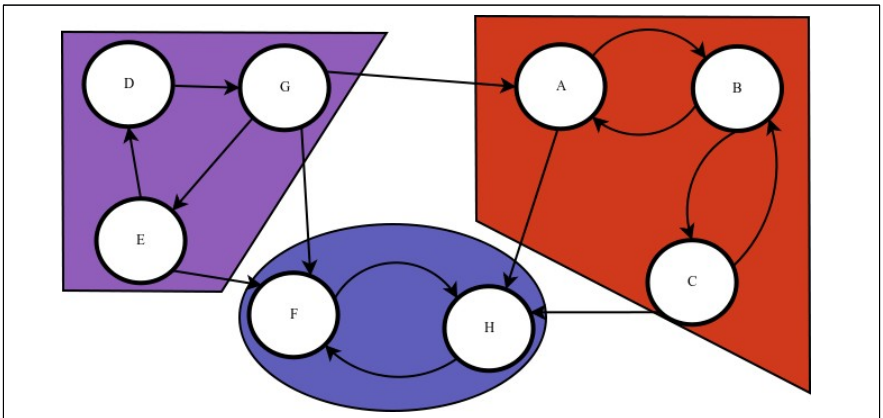
- Adyacencia: dos aristas son adyacentes si tienen un vértice en común. De la misma manera, dos vértices son adyacentes si una arista los une.
- Incidencia: una arista es incidente a un vértice si esta lo une a otro.
- Ponderación: corresponde a una función que en cada arista se le asocia un valor (costo, peso...).
- Etiquetado: dar un nombre a las aristas y/o vértices para distinguirlos.

2.3.3. Definiciones[7,15]

Algunas definiciones necesarias para entender el funcionamiento del programa:

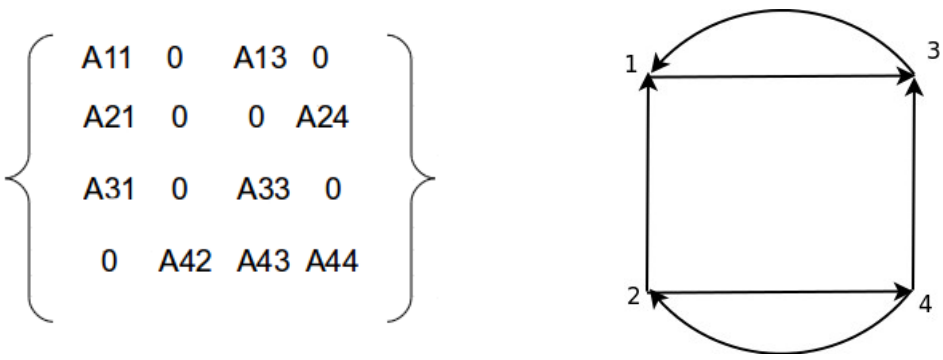
- Camino: un camino uniendo *x* e *y* es una secuencia de aristas $\{e_1,...,e_k\}$ tal que:
  - e_1 es adyacente a *x* por un extremo y a e_2 por el otro.
  - e_k es adyacente a *y* por un extremo y a e_{i-1} por el otro.
  - Cada e_i , $2 <= i <= k -1$, es adyacente a e_{i-1} por un extremo y a e_{i+1} por el otro.
- Conexo: un grafo es conexo si cada par de vértices está unido por un camino.
- Subgrafo: es un grafo obtenido a partir de otro suprimiendo aristas o vértices.
- Fuertemente conexo: es aquel subgrafo que presenta un camino de cada uno de sus nodos a otro[1].
- Componente fuerte: si es fuertemente conexo y no puede ser ampliado sin añadir nodos extra y asociar aristas. Es decir, cada nodo puede pertenecer a un solo componente fuerte. De esta manera un componente fuerte define una partición del grafo (ver Imagen I.1.)[1].

– DFS (*Depht-first search*): la búsqueda en profundidad es un algoritmo que permite recorrer todos los nodos de un grafo de manera ordenada pero no uniforme. Su funcionamiento consiste en ir expandiendo todos los nodos que va encontrando de manera recurrente, de un camino. Cuando ya no quedan más nodos, vuelve atrás, repitiendo el proceso con todos los nodos adyacentes del nodo ya procesado[1].



I.1. Grafo con sus componentes fuertemente conexas remarcadas

Para describir el funcionamiento del algoritmo de Tarjan hay que asociar al sistema de ecuaciones un grafo dirigido, imagen I.2., donde los vértices sean etiquetados con números, por ejemplo de 1 hasta *n*, y unas aristas (i, j) uniendo pares de nodos. La arista (i, j) deja el nodo *i* y apunta hacia el nodo *j*. En la terminología de la teoría de grafos, como hemos comentado antes, serían dirigidos y por tanto no aceptan bucles hacia sí mismos, es decir, aristas (i, i).



I.2. Una matriz y su grafo asociado.

2.3.4. Algoritmo de Tarjan[7,8,9,15]

Se hace una búsqueda *DFS* comenzando de un nodo cualquiera. Las raíces de los componentes fuertemente conexos son las raíces de los subárboles del árbol de búsqueda.

Los nodos se ponen en una pila[1] conforme son visitados. Cuando la búsqueda vuelve de un subárbol, los nodos quedan fuera de la pila y se comprueba si son la raíz de un componente conexo

1 La definición de pila esta en la siguiente pagina, en el apartado 2.4.

o no. Si algún nodo lo es, todos los nodos por encima de él en la pila quedan fuera de esta formando un componente fuertemente conexo.

Para saber si un nodo es raíz o no durante el *DFS* se le asigna un valor *índice* a cada nodo, que es el puesto en el que ha sido visitado ese nodo durante el DFS. Además, se le asigna un valor LowLink el cual es el menor valor entre el índice del nodo v y el nodo v', siendo v' alcanzable desde v. Si LowLink es igual al índice entonces ese nodo es la raíz de un componente fuertemente conexo.

Este algoritmo es capaz de encontrar los componentes fuertemente conexos de un grafo con un número de operaciones de O(n). Es decir del mismo orden de magnitud que el número de nodos que hay en el grafo.*

2.4. Información genérica

Definiciones de palabras que usaremos a lo largo de la memoria:

- MathMl: lenguaje cuyo objetivo es el de presentar notación matemática de forma que distintas máquinas puedan entenderla. Se utiliza en programas como Maple o MathCad.
- Java: lenguaje de programación orientado a objetos y que se ejecuta sobre una máquina virtual, la Java Virtual Machine, por lo que requiere que esta esté instalada.
- Java Virtual Machine: el código de java se compila a bytecode, un lenguaje cercano al de la máquina, y la máquina virtual se encarga de traducir ese bytecode al lenguaje de máquina final.
- Cero de la máquina: es el valor *x* que hace que *1+x=1,* siendo *x* mayor que cero[15].
- Lista: es un vector dinámico, es decir, se puede ampliar o reducir, no se limita a información numérica, puede contener cualquier tipo de datos.
- Pilas: son un tipo de listas en las que se accede únicamente a un elemento. Tipos:
  - Si es el último introducido, de tipo FILO, el primero que entra será el último en salir.
  - Si es el primer introducido, de tipo FIFO, el primero que entra es el primero que sale.
- Residual: es el resultado de sustituir en una función los resultados numéricos. Al no ser exactos, los resultados no valen cero. Sin embargo, se debe acercar mucho. El resultado no puede ser aceptado de otra manera.
- Renderizado de ecuaciones: Escribir las ecuaciones más apropiadamente, es decir, como una imagen y no con representada con caracteres, ejemplo en la imagen I.3.

$$y + x = 5$$

I.3. Ecuación renderizada.

* Más información sobre Tarjan en el anexo 1.

3. Información sobre el programa

3.1. Lenguaje de programación

Para la realización del programa hemos seleccionado Java[12,13,14], como lenguaje de programación debido a los siguientes aspectos:

- Es multiplataforma por naturaleza. No hay que hacer nada especial para que un programa hecho completamente en Java pueda ejecutarse en cualquier plataforma.
- Es un lenguaje de programación menos dado a errores comparado con otros como C/C++ o Fortran, ya que por ejemplo no hay que preocuparse del gasto de memoria.
- Comparado con otros lenguajes equivalentes a Java como C#, Java resultaba preferible debido a que C# es una tecnología privada y por tanto hacerlo multiplataforma es complejo.
- Es un lenguaje libre y que ha sido muy usado para crear programas libres por lo que consideramos que la cantidad de *software* libre escrito en Java sería bastante extensa lo cual podría ser beneficioso para nuestra tarea.
- Aunque Java requiere tener instalada su máquina virtual, está muy extendida, las ultimas cifras la sitúan en 700 millones de ordenadores[15] y en cualquier caso, instalarla es simple y gratuito, solamente hay que ir a la pagina de internet de java* y descargar el archivo.

Por otra parte el lenguaje de Java presenta un inconveniente y es que comparado con C/C++, es más lento, sin embargo, como puede verse en el anexo 4, los tiempos de cálculo resultantes son aceptables.

3.2. Dependencias

El programa del proyecto tiene el nombre de Engineering Suite, acortado eSuite y lo llamaremos así de aquí en adelante.

Nuestro programa utiliza librerías externas para leer las ecuaciones, derivarlas, evaluarlas... Así como también para la representación renderizada de las mismas. Estas librerías deben estar escritas en Java si queremos que la plataforma sea compatible con todos los sistemas operativos del mercado. En la imagen I.4. (véase en la siguiente página) podemos observar un pequeño esquema de las más importantes y de cómo se relacionan.

3.2.1. CommonsMath

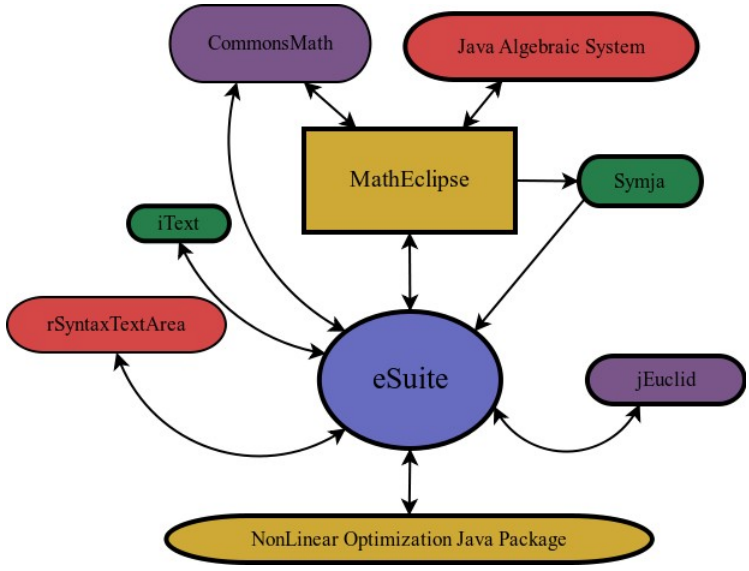
Es un conjunto de librerías realizadas por la fundación Apache donde se recogen métodos

* Página de descarga de Java: http://www.java.com/es/download/

matemáticos que Java no incluye, como por ejemplo trabajar con matrices, derivación, herramientas para el trabajo estadístico, etc.

3.2.2. Java Algebraic System

Son librerías que sirven para trabajar con funciones matemáticas que sí contiene Java.



I.4. Relación entre las principales librerías

3.2.3. jEuclid

Estas librerías se utilizan para el trabajo con el renderizado de ecuaciones. Trabajan con el lenguaje MathMl y las vamos a utilizar para la sección de renderizado, que más adelante explicaremos.

3.2.4. RsyntaxTextArea

Esta librería es un procesador de textos con funciones avanzadas como resaltado de sintaxis en función del lenguaje de programación, marcado de paréntesis, deshacer, rehacer, etc. La utilizaremos tanto para mostrar información, como para que el usuario la introduzca.

3.2.5. MathEclipse

Es un programa de cálculo simbólico escrito en Java. Utiliza las librerías CommonsMath, jEuclid y Java Algebraic System. Es importante remarcar que este programa diferencia entre mayúsculas y minúsculas. Es capaz de leer ecuaciones, evaluarlas, derivarlas, hacer integraciones numéricas, sumatorios, multiplicatorios, sacar valores y vectores propios, traducir ecuaciones al lenguaje MathMl, etc.

Lo hemos implementado de dos maneras:

- Por un lado, como motor interno del programa de resolución de sistemas de ecuaciones, se encarga de leer las ecuaciones, derivarlas y en cada iteración de los algoritmos, evaluarlas en los nuevos puntos.

– Por otro, hemos dado acceso directo al programa desde la sección Mathematics, que más adelante explicaremos.

3.2.6. Symja

Programa de cálculo simbólico basado en MathEclipse. Hemos utilizado algunas de sus librerías para hacer el renderizado de ecuaciones y para las gráficas en la sección de Mathematics.

3.2.7. iText

Son librerías que sirven para trabajar con archivos PDF. Gracias a ellas podremos exportar a PDF.

3.2.8. Nonlínear Optimización Java Package

Librerías para optimizar ecuaciones y sistemas de ecuaciones. Contienen los algoritmos Búsqueda lineal, Double Dogleg, Hebden-Moré y Levenberg-Marquardt.

3.2.9. AlgoWiki

De aquí hemos sacado el código de nuestra implementación del algoritmo de Tarjan.

3.2.10. Iconos

Los iconos del programa se basan en el proyecto Tango.

3.3. Secciones del programa

El programa está dividido en dos secciones principales:

– Solver: esta sección representa el objetivo principal de este proyecto y su función es resolver sistemas de ecuaciones. Está dividido en cuatro secciones o pestañas:

  – Ecuaciones: esta subsección es la que se escriben las ecuaciones a resolver. Incluye resaltado de paréntesis, coloreado de sintaxis, búsqueda, comentarios en distinto color, deshacer, rehacer, cortar, copiar y pegar.

  – Ecuaciones renderizadas: subsección en la que se muestran las ecuaciones renderizadas para facilitar su lectura.

  – Log: esta subsección muestra la información que resulta de resolver el sistema de ecuaciones, concretamente, la aparición de las variables o los residuales de las ecuaciones.

  – Resultados: subsección que comprende las soluciones repartidas en columnas.

– Mathematics: sección que permite interactuar de manera directa con el motor MathEclipse, es por tanto un programa de cálculo simbólico. Incluye también la posibilidad de hacer gráficas.

Como hemos mencionado, MathEclipse diferencia entre mayúsculas y minúsculas, para poder trabajar paralelamente con las dos secciones hacemos que Mathematics trabaje con mayúsculas y Solver con minúsculas. Evitando de esta manera conflictos entre las variables de ambos programas.

4. Funcionamiento del programa

A continuación, se enumeran y analizan las diferentes etapas que podemos apreciar en el proceso de resolución de las ecuaciones, así como sus principales características:

4.1. Lectura y análisis de las ecuaciones*

Los objetivos de esta sección son:

- Almacenar las ecuaciones en una sintaxis que MathEclipse entienda.
- Comprobar que no hay errores en el código escrito por el usuario y si los hay, avisarle.

En todo el proceso de resolución, son necesarias tres listas:

- Funciones: su labor es la de almacenar las ecuaciones y las variables que contiene cada ecuación.
- Variables: contiene todas las variables y aporta información del número de apariciones de cada una de ellas.
- Case Variables: guarda las variables según las introdujo el usuario por primera vez en Solver, en lo que a mayúsculas y minúsculas se refiere.

El almacenamiento interno de las distintas funciones y variables se realizará por medio de cadenas de texto. Esto se debe a que MathEclipse se usa, a nivel interno, como un programa matemático por línea de comandos, es decir, introduciendo órdenes. Se puede apreciar en el siguiente ejemplo: para derivar escribiríamos D[Función,Variable].

Cuando el programa debe resolver una ecuación o un sistema de ecuaciones, la primera tarea es guardar todo el texto en la memoria RAM. Tras esto lo analizamos carácter a carácter buscando los símbolos de los comentarios (recordemos que dichos símbolos son /* */) para que, en caso de haber, podamos localizar y eliminar los comentarios fácilmente. Esto puede resultar insuficiente en determinadas circunstancias, por ejemplo en caso de haber comentarios de varias líneas, pues quedarían eliminados los saltos de línea, pudiendo provocar que dos líneas se junten. Para evitarlo, a la vez que leemos carácter por carácter, cogeremos la cantidad de caracteres que hay en la línea actual y si al llegar a esa cantidad estamos aún en ese comentario, incluiremos un salto de línea. De esta manera conseguimos eliminar los comentarios manteniendo intacto el número de líneas.
Después, eliminamos los espacios y las tabulaciones que hay en todo el texto.
Tras haber suprimido los comentarios, tabulaciones y los espacios siguiendo el método anterior,

* En el anexo 8 hemos incluido un diagrama de flujo para facilitar el entendimiento de esta sección. Y en el anexo 6 todo el código relacionado con la implementación de los algoritmos.

procedemos a buscar llamadas a funciones termodinámicas, que corresponden a la siguiente estructura: Sustancia.Propiedad(Variable1, Variable2…). Para detectar estas llamadas leeremos línea a línea, las ecuaciones buscando algún punto.

Pasado este paso, comprobaremos si la expresión anterior al punto es una sustancia de la base de datos. En caso negativo, la dejaremos intacta, sin embargo, si corresponde con un material, procedemos a dividir la linea según puntos, paréntesis y comas. Gracias a esta división, podemos analizar la información resultante de la siguiente manera: antes del punto aparecerá la sustancia, después del punto, la propiedad y antes de las comas, las variables, exceptuando la ultima variable.

Una vez que tengamos toda la información, comprobaremos que la sustancia tiene almacenada esa propiedad, si no es así, pararemos la ejecución avisando al usuario del problema. Por otra parte, si el resultado de la comprobación es positivo, procedemos a cambiar la llamada a la función termodinámica por la fórmula correspondiente, guardada en la base de datos. Utilizaremos las variables introducidas por el usuario para sustituir en la fórmula almacenada las variables antiguas por las nuevas, por ejemplo la fórmula almacenada puede contener como variables, temperatura y entalpía pero el usuario puede preferir llamarlas Temp_dia y Ental_dia.

A modo de resumen, hasta el momento hemos eliminado los comentarios, las tabulaciones. los espacios y las llamadas a funciones termodinámicas. Por tanto como solo nos quedan ecuaciones, es el momento de realizar una traducción de la sintaxis de eSuite a la sintaxis de MathEclipse, mucho más restrictiva. Para ello, es fundamental establecer una pequeña aclaración sobre las características fundamentales de cada una:

Sintaxis de MathEclipse:

- Después de las funciones Sin, Cos, Tan, ArcTan, ArcCos, ArcSin, Sinh, Cosh, Tanh, Exp y Log es preciso que la información vaya entre corchetes. Estas funciones deben estar escritas siguiendo los patrones anteriores.
- Es sensible a las mayúsculas y minúsculas. Por lo que $x \neq X$ **(5)**
- Solamente permite introducir puntos para los decimales.
- Trabaja con los siguientes símbolos: a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0 + - * / , . ^ ( ) [ ] =
- No acepta ecuaciones, solamente funciones. Es decir, no aceptaría por ejemplo $x+x^2=2$ **(6)** en cambio, sí aceptaría $x+x^2-2$ **(7)**. Solamente acepta el signo = para definir variables, es decir $x=2$ **(8)** sí lo aceptaría.
- Rechaza variables que empiecen por números y que contengan símbolos distintos a

letras o números.

- Solamente trabaja con Radianes con una excepción, pues contempla una constante, Degree, para pasar de radianes a grados fácilmente.

Sintaxis de eSuite:

- No hay diferencia en el uso o incluso la alternancia, entre paréntesis y corchetes.
- No discrimina entre mayúsculas y minúsculas.
- Comas y puntos pueden servir para marcar los decimales indistintamente.
- Debe permitir el símbolo _, aparte de los de MathEclipse, ya que es muy frecuente a la hora de nombrar variables. Por ejemplo, "Agua_Salada". De otra manera, sería necesario incorporar las dos palabras unidas, sin espaciado, con la dificultad que conlleva para su comprensión y lectura.
- Solamente acepta ecuaciones.
- Rechaza variables que empiecen por número y que contengan símbolos distintos a letras y números.
- Trabaja tanto con radianes como con grados.
- La información debe estar estructurada de tal manera que no se acumule más de una ecuación por línea.

Conforme se realiza la traducción de las ecuaciones, se comprueba que no haya errores sintácticos o fallos en la escritura y se sacan las variables de las ecuaciones por medio de una lectura exhaustiva, línea por línea, con un meticuloso análisis del contenido. Esto se realiza en dos fases:

4.1.1. Análisis de caracteres

Mediante la lectura carácter a carácter se va corrigiendo todo para reducirlo a minúsculas. En caso de encontrar un _ se transforma a Gg. Provocará indecisión al sospechar que puede resultar problemático junto a una hipotética variable Gg, pero como todas las variables introducidas por el usuario van a ser traducidas a minúsculas, unas no interferirán con otras.

Las comas, que deben aparecer entre números, se transformarán en puntos. En caso de no aparecer entre números, es preciso parar la ejecución avisando del problema y marcando su localización.

También se comprueba que no haya funciones trigonométricas, hiperbólicas, logaritmos ni exponenciales que no tengan ninguna información asociada. Es decir, si se lee Sin() pararíamos la ejecución para avisar al usuario del tipo de error, función vacía en este caso y remarcando la fila en la que se encuentra.

Si hay dos operadores seguidos, exceptuando + y - se lanza un error y se avisa al usuario, señalando dónde ha ocurrido.

Paralelamente, comprobaremos que no queda ningún operador vacío. Por ejemplo, que no encontremos un símbolo de multiplicar al final de la ecuación o antes de paréntesis o corchetes de cierre, lugares que no corresponden a dichos elementos. Si ocurre, se avisa al usuario, de la misma manera que para los errores anteriores.

Para transformar la ecuación en función, es necesario sustituir el símbolo = por -1*(. Al final de esa fila cerramos el paréntesis. De esta manera, pasamos de una ecuación como $x=45$ **(9)** a una como $x-1*(45)$ **(10)**.

Antes de pasar a la siguiente fase, es imprescindible comprobar que hay un signo =.

4.1.2. Análisis por cadenas de texto

El principal objetivo de este segundo paso es sacar las variables de la ecuación y traducir los paréntesis y corchetes introducidos por el usuario a la sintaxis de MathEclipse. Para ello, seccionaremos la ecuación según aparezcan los siguiente símbolos: + / * - ( ) [ ] ^

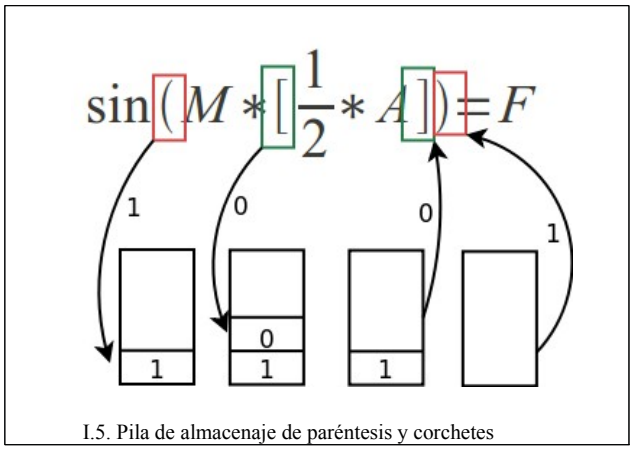
No obstante, haremos también algunos cambios en la información que nos llega para adaptarla.

Una vez dividida la ecuación siguiendo los patrones anteriores, iremos analizando las subcadenas obtenidas, las cuales, al haberlas separado en función de los operadores, deberían obtener variables, números constantes -pi o e- o funciones trigonométricas, hiperbólicas, exponenciales y logarítmicas.

Para traducir los paréntesis o corchetes introducidos por el usuario a la sintaxis de MathEclipse, que requiere corchetes tras las funciones mientras que para todo lo demás hace uso de paréntesis, los introduciremos en una pila de tipo FILO, *First In Last Out,* de variables binarias, la cual utilizaremos de la siguiente manera:

Si encontramos un paréntesis o corchete de abrir, introduciremos un elemento en la pila. Si, por el contrario, nos llega un paréntesis o corchete de cerrar, sacamos un elemento de la pila. Para ir introduciendo elementos, distinguiremos con un 1 o un 0; si el paréntesis o corchete va justo detrás de una función, introduciremos un 1 y traduciremos la entrada a un corchete. Si no es una función, introduciremos un 0 y escribiremos un paréntesis.

A la hora de ir extrayéndolos, procederemos de la misma manera: si nos sale un 0, sustituiremos el paréntesis o corchete por un paréntesis y si nos sale un 1, escribiremos un corchete de cerrado. Tenemos como ejemplo de funcionamiento la imagen I.5.



I.5. Pila de almacenaje de paréntesis y corchetes

Es fundamental transformar la subcadena ln a log para permitir trabajar con el logaritmo natural de otra manera y no solamente con log.

Si nos llega cualquier función, la traducimos a su forma oficial para MathEclipse. Por ejemplo, cos lo traduciremos a Cos. Si está activado el uso de grados en vez de radianes, el contenido de las funciones trigonométricas lo envolvemos dentro de Degree(). Por ejemplo:

$$\sin(x) \rightarrow \sin[Degree(x)] \quad \textbf{(11)}$$

Aunque MathEclipse posee la constante Pi, da problemas utilizarla y por ello, la traducimos por el valor numérico que Java tiene almacenado.

Para sacar las variables, la mejor manera es hacerlo por descarte: será una variable cuando no corresponda a un número, ni una función, ni un símbolo matemático. Al sacar una variable comprobaremos que no empiece por un número, en cuyo caso, pararíamos la ejecución del programa avisando del error. En cambio, si todo es correcto, almacenaremos las variables en una lista temporal.

Al acabar de leer la ecuación, es primordial comprobar que la pila de los paréntesis o corchetes esté vacía, es decir, que haya tantos paréntesis o corchetes de abrir como de cerrar. En el caso de que estén descompensados, avisaremos al usuario del error. Tras esta comprobación, guardaremos la ecuación traducida con las variables que contiene en la lista de Funciones.

En otra lista, la de las variables, comprobaremos si las variables de esa ecuación, almacenadas en la lista temporal, estaban ya recopiladas. Las almacenaremos en caso de que no apareciesen. En cualquier caso, aumentaremos en uno las veces que aparecen las correspondientes variables.

De esta manera vamos introduciendo la información en las listas de funciones y de variables.

Como las variables están en minúsculas y queremos presentarlas al usuario como él las escribió, hacemos una traducción menos severa, paralelamente a la que acabamos de explicar, en la que

únicamente eliminaremos el signo =, las comas, y _. Del mismo modo, transformaremos todos los corchetes en paréntesis. Podemos pedirle a MathEclipse que nos devuelva las variables, que habrán sido guardadas en una lista llamada Case Variables. Estas variables sí que mantienen la información de mayúsculas y minúsculas y son las que utilizaremos para mostrar al usuario. Sin embargo, solamente almacenamos la información en función de cómo fue introducida la variable por primera vez. Como consecuencia de esta traducción de menor calidad, puede suceder que interprete ciertos datos, por ejemplo, "cos", como una variable. Para ello simplemente aplicaremos un filtro en el que si una variable es en realidad una función, no la guardará en la lista.

Para finalizar esta etapa, comprobaremos que el número de ecuaciones y variables coincide. De no ser así, se avisa al usuario del problema y se le recuerda que en la sección Log puede comprobar el número de apariciones de las distintas variables.

4.2. Procesamiento de las ecuaciones

En este punto dispondremos de todas las ecuaciones debidamente formateadas para funcionar en MathEclipse con sus variables, lista de funciones, así como una lista con todas las variables, lista de variables.

Para la resolución numérica es muy importante que el sistema de ecuaciones sea lo menor posible, ya que cuantas más ecuaciones haya en el sistema, más posibilidades hay de que el algoritmo falle. Por tanto nos interesa reducir los sistemas de ecuaciones al mínimo, porque aunque haya por ejemplo 50 ecuaciones eso no quiere decir que el sistema sea de 50 x 50. Puede ser que sean 50 ecuaciones independientes o que sean varios subsistemas. Este ha de ser nuestro siguiente objetivo y lo dividiremos en dos etapas, la resolución de todos los sistemas de una sola ecuación y la utilización del algoritmo de Tarjan.

4.2.1. Resolución de todos los sistemas de una sola ecuación

En primer lugar, realizamos una búsqueda en la lista de funciones buscando ecuaciones que contengan solamente una variable, si encontramos alguna, la resolveremos. Al ser ecuaciones de una variable, primero comprobaremos si es una constante, mediante el método de Newton-Raphson para una variable pero sin llegar a derivarla ya que, si es una constante, su derivada debe valer 1. Entonces, si es una constante, nos habremos evitado hacer una derivada y aplicar un algoritmo complejo. Sin embargo, si no lo es, pasaremos la ecuación al método apropiado para que quede resuelta.

Una vez solucionada, avanzaremos en tres direcciones: por un lado, pasaremos esa función a una

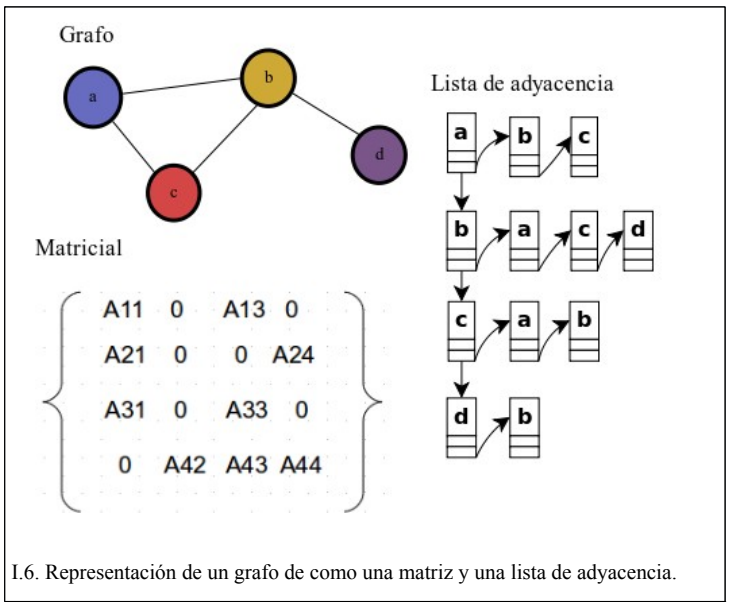
lista de funciones resueltas; por otro lado, de la misma manera, insertaremos la variable correspondiente a una lista diferente de la de variables; para terminar, eliminaremos la variable de todas las funciones, pues cada ecuación guarda sus variables. Entonces, hemos eliminado variable y función de las listas globales; ahora esa variable recibirá el trato de una constante en las demás funciones. Esto conlleva que ecuaciones que antes contuvieran dos variables, siendo una de ellas la que ha quedado resuelta, ahora tendrán únicamente una incógnita. Repetiremos la búsqueda hasta que ya no queden más.

4.2.2. Utilización del algoritmo de Tarjan

Llegado este momento, solamente quedan sistemas de ecuaciones o ecuaciones que tienen valores que dependen de esos sistemas de ecuaciones. Para sacar los subsistemas procederemos a utilizar el algoritmo de Tarjan.

Para utilizar este algoritmo, lo primero es crear un grafo de las variables y las funciones, debido a que solamente trabaja con grafos. Para ello nos vamos a servir de una lista de adyacencia. Para facilitar su comprensión hay una matriz como grafo y lista de adyacencia en la imagen I.6.

Crearemos un nodo por variable y un nodo por función. Es importante ahorrar memoria, de tal manera que llamaremos a estos nodos por la posición que ocupen en las listas globales asignando valores positivos para las variables y negativos para las funciones. Esta distinción es importante ya que el algoritmo de Tarjan simplemente devuelve una lista con estos números, entonces, utilizaremos el signo para discernir si es función o variable.



I.6. Representación de un grafo de como una matriz y una lista de adyacencia.

Una vez creados estos nodos, elaboraremos la lista de adyacencia haciendo las aristas o relaciones entre las funciones y las variables que contienen esas funciones recorriendo las funciones y mirando sus variables.

Una vez que tengamos la lista de adyacencia, deberemos llamar al algoritmo de Tarjan. Para utilizar este algoritmo, aparte de la lista de adyacencia, es necesario introducir un punto inicial. Escogemos la variable con menos apariciones en la lista de adyacencia[2]. Si hay varias, introduciremos una, guardaremos el resultado, y así con todas para al final, analizar cuál es mejor y quedarnos con ese resultado.

Una vez aplicado el algoritmo nos devuelve una lista con números, son las funciones y variables, negativos y positivos. Procedemos a crear una matriz de relaciones, es decir, una matriz con las funciones en las filas y las variables en las columnas, asignando un 1 si en esa función está esa variable o un 0 si no lo está. Como puede haber varias variables con el mismo número de apariciones ejecutaremos Tarjan para cada una de ellas y nos quedaremos con la mejor solución. Para comprobar qué solución es mejor, contamos el número de unos que hay por debajo y en la diagonal. Seleccionaremos la solución que tenga más unos en la diagonal y debajo de esta.

Para explicar mejor todo este proceso y entender por qué queremos que el resultado sea diagonal inferior vamos a utilizar un ejemplo:

$$2*x+y=1 \qquad \textbf{(12)}$$

$$a+b=x \qquad \textbf{(13)}$$

$$x*y=3 \qquad \textbf{(14)}$$

$$3*a*b=6 \qquad \textbf{(15)}$$

Podemos observar que en principio es un sistema de 4 ecuaciones con 4 variables.

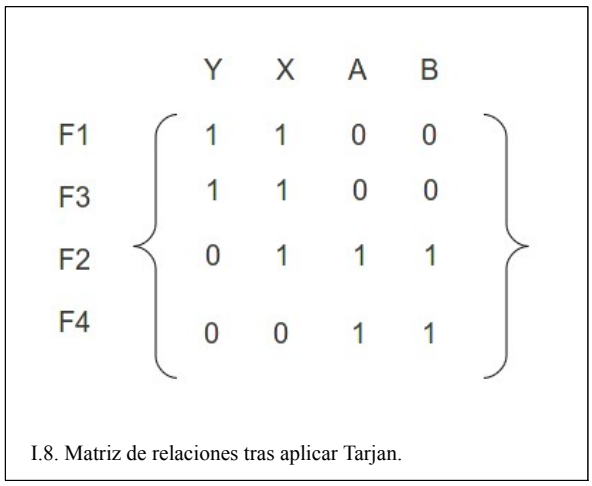
Este sistema antes de aplicar Tarjan se vería de la siguiente manera en una matriz de relaciones, imagen I.7:

	X	Y	A	B
F1	1	1	0	0
F2	1	0	1	1
F3	1	1	0	0
F4	0	0	1	1

I.7. Matriz de relaciones del ejemplo.

Después de aplicar Tarjan introduciendo B como punto inicial, tenemos la matriz de relaciones de la imagen I.8:

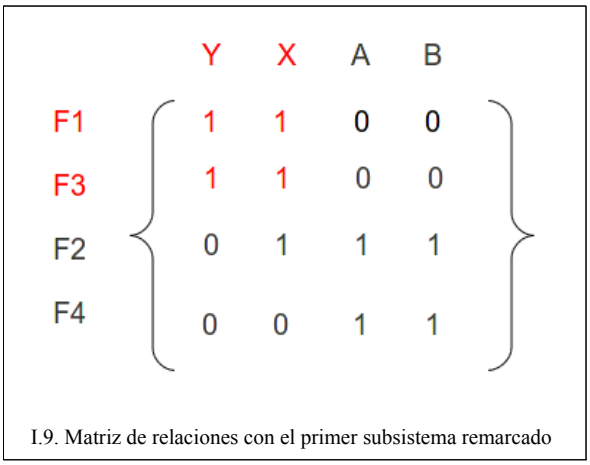
2 Esta metodología la hemos sacado de los puntos 8 y 9 de la bibliografía.



I.8. Matriz de relaciones tras aplicar Tarjan.

Como podemos observar, F2 y F3 han cambiado sus posiciones así como X e Y.

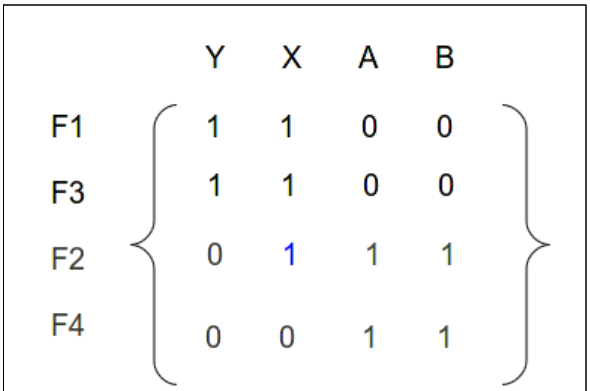
Una vez realizado este proceso, pasamos a buscar subsistemas de ecuaciones analizando línea a línea la matriz de relaciones. Vamos a explicar el proceso continuando con el ejemplo anterior. Leemos la primera fila y vemos que hay dos variables, X e Y pero solamente una función. Por tanto, no se puede resolver el sistema y avanzamos a la siguiente fila, leemos la siguiente y vemos que tiene dos variables, X e Y. Las variables coinciden y tenemos ya dos funciones por lo que sacamos un sistemas de ecuaciones. En la imagen I.9. está remarcado en rojo el primer subsistema de ecuaciones.



I.9. Matriz de relaciones con el primer subsistema remarcado

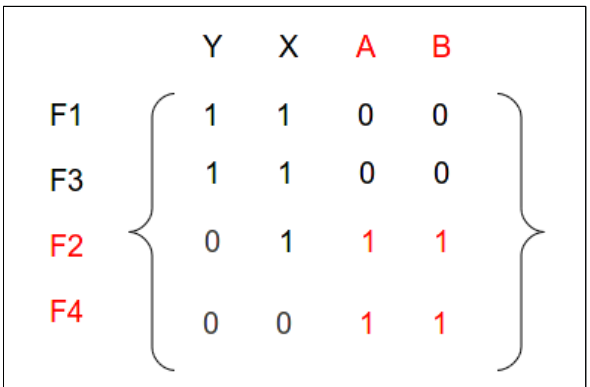
Si por el contrario hubiese salido una variable más, se hubiese continuado el proceso hasta que todas las variables coincidiesen y además el número de filas leídas y variables coincidiese.

Continuando con el ejemplo, vemos que hemos resuelto X e Y gracias a las dos primeras ecuaciones así que esas dos variables las damos por resueltas y las trataremos a partir de ahora como constantes De hecho, en el programa eliminamos las filas y columnas correspondientes de la matriz de relaciones, pero aquí, para que sea más visible, no lo vamos a hacer en el ejemplo. Por lo que la variable marcada en la imagen I.10. es una constante.



I.10. Matriz de relaciones con la variables de conexión coloreada.

Hemos analizado las dos primeras filas. Pasamos a la tercera, que únicamente consta de dos variables, A y B, ya que X está resuelta. Como son dos variables en una sola función, avanzamos a la siguiente fila. Al coincidir las variables ya tenemos nuestro segundo subsistema, como queda reflejado en la imagen I.11.



I.11. Matriz de relaciones con el segundo subsistema remarcado.

Nótese que esta manera de analizar la matriz de relaciones nos garantiza que, en el peor de los casos, si Tarjan no consiguiese diferenciar los subsistemas o pusiese una función que depende de una variable que está más abajo en la matriz, se podría resolver aunque nos quedase un sistema de ecuaciones innecesariamente grande.

Habrá que repetir esta metodología por casa sistema de ecuaciones independiente.

4.3. Resolución de las ecuaciones

En la resolución de ecuaciones, utilizamos métodos numéricos iterativos, es decir, que repitan las mismas operaciones hasta que encuentren el resultado. Hemos empleado los métodos implementados en la librería *Nonlínear Optimización Java Package.*

Esuite tiene implementados 4 algoritmos de resolución que se subdividen en dos tipos:

– Librería Uncmin_f77: estos son el algoritmo de Búsqueda lineal, Double Dogleg y Hebden-Moré. Estos algoritmos son métodos de optimización así que, para que resuelvan nuestro

sistema de ecuaciones se trata con ellos mediante una función objetivo de la siguiente forma:

$$f = \frac{1}{2}\sum F^2 \quad \textbf{(16)}$$

Hay que introducirles en cada iteración el valor de **(16)** en el punto además del valor del gradiente de **(16)**, vector que contiene las derivadas de f respecto a sus variables, y el de la Hessiana de **(16)**, matriz que contiene las derivadas del gradiente respecto a todas las variables. Para ahorrar tiempo de cálculo y número de evaluaciones, utilizaremos las siguientes fórmulas para calcular el Gradiente y la Hessiana, las cuales únicamente sirven debido a la forma peculiar de **(16)**:

Gradiente: $G = J * F$ **(17)**

Hessiana: $H = J' * J + \sum Fj + \nabla^2 Fj$ **(18)**

Siendo F = Vector de las funciones.

J = Jacobiana de F.

J'= Jacobiana transpuesta.

∇^2 = Operador de segundas derivadas.

– Levenberg-Marquard: este algoritmo es más sofisticado ya que incluye escalado de las funciones para evitar problemas con el cero de la máquina. Como este método crea internamente la función objetivo **(16)**, solamente requiere de las ecuaciones evaluadas y de la Jacobiana.

Nótese que todos los algoritmos de resolución son en realidad métodos de optimización por lo que en realidad se está buscando un óptimo y no la solución del sistema. Esto puede causar que el algoritmo devuelva en vez de la solución, un punto óptimo. Si esto ocurre, se avisará al usuario diciéndole que debe probar un punto inicial distinto o que pruebe usando un algoritmo diferente.
Para los algoritmos de la librería Uncmin_f77 se ha incluido un algoritmo al final que en el caso de que el algoritmo devuelva un óptimo que no sea la solución, aplique un número de iteraciones, especificadas por el usuario del método de Newton-Raphson. En ese caso, si está en un punto que no es la solución, este método da un salto muy grande alejándose del punto óptimo. Como en un primer término, esto resulta inestable -recordemos que el método de Newton-Raphson sin control de paso solamente tiene convergencia local- está desactivado permitiendo al usuario que lo active si lo considera oportuno.

Todos estos métodos requieren un punto inicial. Se ha tomado como punto inicial por defecto el uno. Esta elección se debe a que es un número para el cual todas las funciones existen, por ejemplo $\log(1)$ **(19)** y $\arctan(1)$ **(20)**. Sin embargo, se permite al usuario cambiar este punto inicial o cambiar el punto inicial de cada variable independientemente; algo muy importante porque debido a

la naturaleza de la resolución numérica, solamente se devuelve una solución de las ecuaciones. Por ejemplo, en el caso de: $x^2=1$ **(21)** X puede valer o 1 o -1. Si el punto inicial es -2 el resultado de **(21)** será -1 sin embargo si es 2 el resultado será 1.

4.4. Mostrar los resultados al usuario

Al acabar la resolución de los sistemas mostramos al usuario la siguiente información:

- Resultados: los valores encontrados por los algoritmos.
- Residuales: se muestran las ecuaciones evaluadas en la solución y si se dan residuales por encima de la raíz cuadrada del cero de la máquina, se avisa al usuario de que hay residuales elevados.
- Número de apariciones de las variables.

Para mostrar los resultados, residuales y apariciones de las variables utilizamos la lista que habíamos creado inicialmente para contener la información de mayúsculas y minúsculas. Además de transformar Gg a _.

Los residuales y las apariciones se muestran en la sección Log, mientras que los resultados aparecen en el apartado de resultados.

4.5. Renderizado de ecuaciones

El renderizado de ecuaciones se corresponde con la sección renderizado del programa.

El objetivo de esta función es escribir las ecuaciones de una forma más formal para facilitar su lectura.

Se realiza de manera independiente a la resolución de las ecuaciones, por lo que el usuario debe pedir explícitamente esta operación.

Para llevar a cabo este método, lo primero que realizamos es la transformación de la sintaxis de eSuite a la de MathEclipse de la misma manera que hacíamos para resolver las ecuaciones. Después, convertiremos esas funciones en ecuaciones. Esta doble transformación nos sirve para adaptar la sintaxis a la de MathEclipse aprovechando el código explicado en el apartado 4.1.

Si hay llamadas a funciones termodinámicas las transformamos a la forma:

Función(Propiedad)=(variable 1,Variable 2...)

Como MathEclipse no acepta comas, las transformamos a una secuencia de caracteres larga y que sea poco probable que el usuario use. Para después buscar esa secuencia y sustituirla por una coma. Debemos proceder de manera equivalente con el símbolo _. Tras tener las ecuaciones con la sintaxis de MathEclipse solicitamos la transformación en formato MathMl. Al realizar esta traducción,

MathEclipse elimina todos los signos “*” por lo que sustituimos antes de pedirle que nos haga la traducción los signos “*” por Xx. Este cambio causa algunos inconvenientes, ya que Xx es en realidad una variable, por lo que para minimizar los errores, eliminamos todos los signos * al lado de los paréntesis. Tras la conversión, sustituimos todas las Xx por *. Recordemos que no hay problema al sustituir todas las Xx por * ya que al realizar la conversión inicial todas las variables están escritas con minúsculas. Así como utilizamos las variables de la lista Case Variables para mostrar las variables tal y como las escribió el usuario, es decir con información de mayúsculas y minúsculas. Finalmente para poder mostrar al usuario las ecuaciones utilizamos las librerías del programa jEuclid, especializado en escribir fórmulas en formato MathMl, para que escriba la fórmula en función del código que tengamos.

4.6. Funcionamiento de eSuite Mathematics

Esta sección es un programa de cálculo simbólico. Aprovechando que teníamos un programa de cálculo simbólico incluido, MathEclipse, le hemos dado acceso directo desde esta parte. Como eSuite Solver trabaja con minúsculas, esta sección hará uso de las mayúsculas para impedir que la asignación de un valor a una variable afecte a las de eSuite Solver.

MathEclipse representa una dificultad y es que utiliza la letra D para hacer derivadas, por lo que si utilizásemos esta letra como una variable, MathEclipse ya no es capaz de hacer derivadas. Para evitar este problema, siempre que se quiera introducir la letra D aislada, se transforma a Dd. De la misma manera, las derivadas se realizan con el comando Deriv en vez de D. Lógicamente el cambio es interno y el usuario no lo aprecia.

En esta sección no hacemos una transformación tan profunda como en Solver por lo que la sintaxis es la misma que en MathEclipse, la única diferencia es que aquí es insensible a mayúsculas o minúsculas.

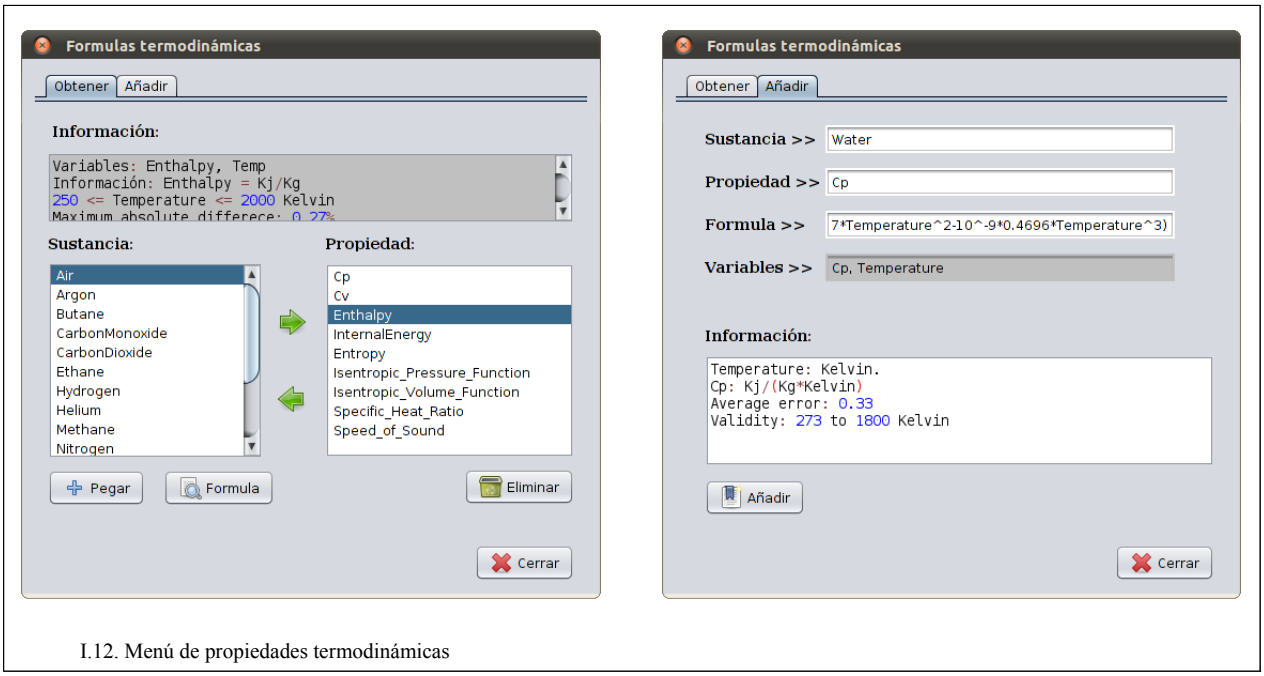
Características:

- Historial: almacena los últimos 200 comandos introducidos.
- Ans: se almacena la última operación realizada en Ans para poder trabajar con ella.
- Renderizado: se puede escoger entre varias opciones: que escriba renderizado los resultados o que los escriba con caracteres normales.
- Ayuda: hemos escrito una ayuda que explica y da un ejemplo de cómo se usa el programa.
- Gráficas: tiene la capacidad de hacer gráficas bidimensionales de varias funciones a la vez.
- Coloreado de sintaxis.

5. Menús del programa

5.1. Propiedades termodinámicas

Desde este menú se pueden añadir llamadas a propiedades termodinámicas a nuestras ecuaciones o podemos ampliar la base de datos añadiendo nuestras propias fórmulas.



I.12. Menú de propiedades termodinámicas

Como podemos apreciar en la imagen I.12. el menú esta dividido en dos pestañas:

- Obtener:
  - Desde Borrar podemos eliminar sustancias o propiedades de la base de datos.
  - El botón Pegar incluye una llamada a la función termodinámica seleccionada actualmente en las ecuaciones, además de la información relacionada.
  - Con Fórmula añadimos la ecuación que da la aproximación a la propiedad.
  - Al seleccionar una propiedad se muestra su información en el campo habilitado para ello.
- Añadir: podemos añadir sustancias nuevas y más propiedades a sustancias ya existentes. Para ello se debe incluir la fórmula de aproximación ya que eSuite no trabaja haciendo interpolaciones entre valores de una tabla.

Como parte del proyecto hemos añadido* la capacidad calorífica, la entalpía y la entropía de las siguientes sustancias: Argón, Butano, Monóxido de Carbono, Dióxido de Carbono, Etano,

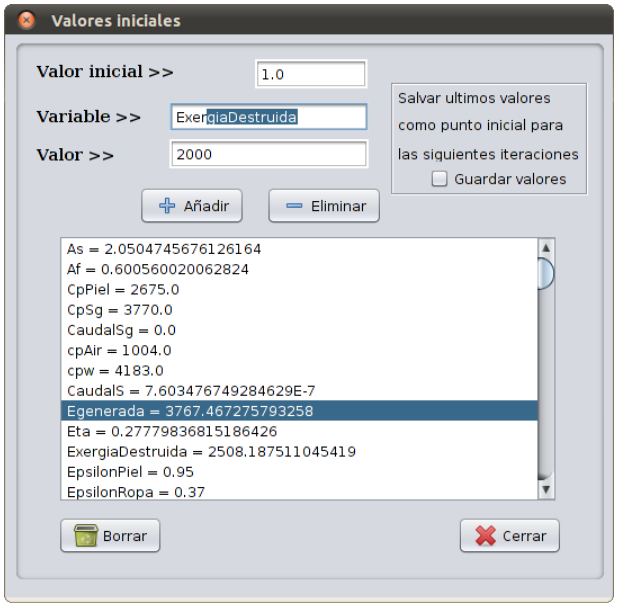
* En el anexo 7 se encuentran todas las formulas incluidas, con sus rangos de validez y unidades.

Hidrógeno, Helio, Metano, Nitrógeno, Oxígeno, Propano y Dióxido de azufre. Para el aire hemos incluido la entalpía y la entropía la capacidad volumétrica, la energía interna, la velocidad del sonido, la presión isotrópica y el volumen isotrópico, aparte de la capacidad calorífica. En el caso del agua la hemos divido en dos, agua saturada y vapor sobrecalentado. Para el agua saturada hemos incluido la capacidad calorífica, la temperatura de saturación, el volumen específico líquido, el volumen específico del vapor, la entalpía para el agua líquida, la entalpía para el vapor, el calor de vaporización, la entropía para agua líquida y la entropía para el vapor. Por otra parte para el vapor sobrecalentado hemos introducido la entalpía, la entropía y el volumen específico. En todas las propiedades hemos incluido las unidades, los límites en los que tienen validez y su desviación o error respecto a los valores reales. Algunos de estos parámetros están divididos en tramos, es decir, para un rango de temperaturas o presión se tiene que escoger una fórmula y sino, otra. Por tanto es el usuario quien debe escoger un intervalo u otro a la hora de calcular.

Todas estas ecuaciones son reversibles. Por ejemplo, se puede usar la formula de la entalpía tanto para calcular la entalpía a una determinada temperatura y presión como se puede calcular la temperatura para una determinada entalpía y presión.

5.2. Valores iniciales

Este es el menú desde el cual trabajamos con los valores iniciales. Podemos cambiar el valor inicial por defecto, el de cada variable o podemos decirle al programa que guarde los últimos valores calculados como iniciales de la siguiente iteración. En la imagen I.13. podemos ver el menú con un ejemplo.



I.13. Menú de valores iniciales.

Además se ha incluido Autocompletado en el campo de variable para evitar que el usuario se equivoque al escribir el nombre de una variable.

5.3. Preferencias

Este epígrafe se divide en tres secciones:

- General.
- Ecuaciones.
- Matemáticas.

5.3.1 General

Desde aquí podemos modificar las preferencias que no tienen que ver con aspectos puramente matemáticos o propios de la sección Mathematics, como se aprecia en la imagen I.14.

Estas preferencias son:

- Trigonometría: podemos escoger si queremos que Solver trabaje con grados o radianes.
- Fuente: podemos cambiar la fuente de la sección de ecuaciones y el tamaño de la fuente en ecuaciones, log y resultados.
- Cambiar el lenguaje a la aplicación, incluimos inglés y español. Es importante mostrar que requiere reiniciar la aplicación.
- Cambiar el tema: podemos escoger entre tres temas independientes del sistema operativo o uno que se adapta al aspecto de este.

I.14. Menú Preferencias\General

5.3.2. Ecuaciones

Nos permite variar parámetros relacionados con los algoritmos de resolución:

- Precisión: cuando la diferencia normalizada entre un paso y el siguiente es menor que este valor se da por bueno el resultado. Por defecto, la raíz cuadrada del cero de la máquina.
- Máximo número de iteraciones: si cuando pasan más iteraciones de las que aparecen en estas líneas, el algoritmo se para avisando al usuario.
- Tiempo máximo de cálculo: es el tiempo que se le permite al algoritmo para resolver un sistema de ecuaciones. Si lo supera el programa para y se avisa al usuario.
- Permitimos escoger qué método queremos utilizar para resolver ecuaciones de una variable y de varias.
- Salto máximo: los algoritmos Búsqueda lineal, Double Dogleg y Hebden-Moré tienen un parámetro que limita un salto entre dos iteraciones seguidas. Desde aquí podemos seleccionar el valor. Por defecto, está puesto un valor elevado para no limitar a los métodos.
- Precisión del gradiente: los métodos Búsqueda lineal, Double Dogleg y Hebden-Moré paran si el gradiente vale menos que el valor aquí dado.
- Evitar mínimos: podemos escoger el número de iteraciones que queremos que se realicen del algoritmo de Newton-Raphson después de los métodos Búsqueda lineal, Double Dogleg y Hebden-Moré, para evitar que caigan en un mínimo. Por defecto, aparece desactivado.
- Región de confianza: es el radio de confianza inicial para todos los métodos, salvo búsqueda lineal. Por defecto -1, lo que significa que es el algoritmo el que decide.

5.3.3. Matemáticas:

Podemos escoger los limites máximos de la sección de hacer gráficas así como el tiempo máximo de cálculo.

5.4. Ayuda

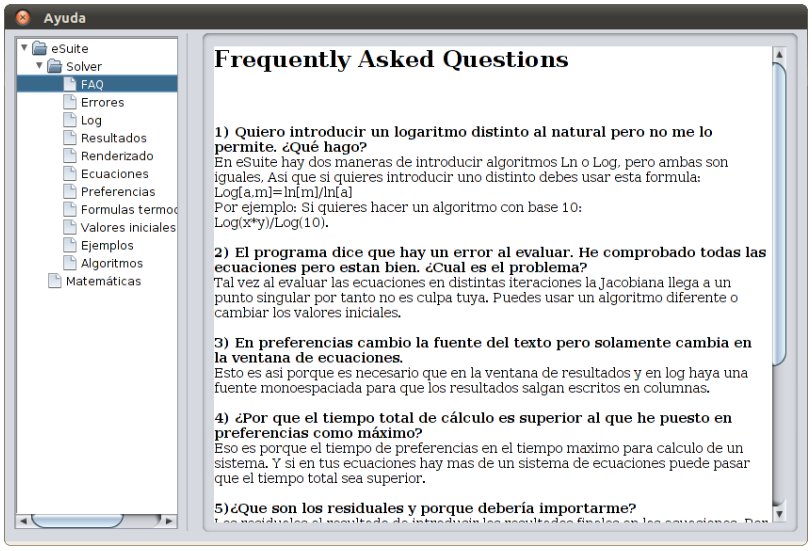
El menú que observamos en la imagen I.16, es imprescindible para solventar las distintas dudas o problemas que se le planteen al usuario. No solamente trata de explicar cómo usar el programa sino también cómo resuelve las ecuaciones para que así, la persona sea capaz de entender los resultados más fácilmente*.

Esta dividido en dos secciones principales:

- Solver: tiene submenús donde se explican los distintos errores posibles, cómo funciona internamente el programa y algunos ejemplos de su utilización.

* En el anexo 6 puede consultar toda la ayuda del programa.

– Matemáticas: explica algunas diferencias sintácticas respecto a Solver.



I.15. Menú de ayuda

Además del menú de ayuda hemos incluido una ayuda inicia, para la primera vez que se ejecute el programa. Si el programa detecta que es la primera vez en ser ejecutado en una máquina, se carga un texto en Equations que incluye información sobre cómo cambiar el lenguaje, cómo cambiar el tema de la aplicación, cómo resolver ecuaciones, cómo renderizarlas. De la misma manera, presenta dos ejemplos, uno con todos los operadores y varias funciones: un seno, un logaritmo y una función hiperbólica, y el otro con una llamada a una función termodinámica de la base de datos.

La sección Mathematics tiene su propia ayuda que sale al escribir el comando”QuickHelp”, ayuda pequeña, o “Help”, ayuda larga; esta ayuda consiste en una lista con los comandos que tiene el programa, una pequeña explicación y un ejemplo.

6. Otras funciones

6.1. Guardar documentos, configuración y base de datos de sustancias

Para todos estos archivos se ha utilizado el formato de texto plano. Para guardar documentos se le ha puesto el formato de archivo .ris y para el de configuración y base de datos .txt.

Como para guardar los documentos se ha utilizado el formato de un archivo de texto plano, este documento puede ser abierto por cualquier editor de texto, permitiendo así que sea más fácil compartir documentos ya que no es necesario el programa eSuite para leer las ecuaciones.
En este archivo se guarda toda la información que hay en Solver/ecuaciones así como los valores iniciales establecidos para las distintas variables.

6.2. Exportar a PDF e imprimir

Para exportar a PDF, el programa incorpora la librería iText. El proceso será sencillo: pasaremos los distintos textos para que los copie en un archivo, luego preguntaremos donde quiere guardar el archivo y finalmente que secciones quiere incluir en el archivo.
En el caso de imprimir acudiremos a *DocumentRenderer,* que actúa igual que iText. En este caso, el usuario podrá apreciar dos fases: primero una ventana preguntándole qué secciones quiere imprimir y luego una segunda, donde deberá escoger las diferentes configuraciones desea aplicar a su impresión: número de copias, calidad, etc.
Para evitar problemas con el ancho de la página, los resultados aparecerán en una sola columna, en vez de en varias.

6.3. Traducción

PPara llevar a cabo la traducción hemos creado una lista que contiene los textos del programa. Luego accedemos a esta lista para mostrar el texto. Esta lista se carga al principio y dependiendo del lenguaje escogido se le asignan unos valores u otros.
Para añadir un nuevo lenguaje seguiremos los siguientes tres pasos:

- Copiar el texto de traducción dentro del archivo Translate.java y crear una nueva condición, es decir, ante if(entrada = español) en el lugar de español, añadiremos un nuevo idioma.
- Traducir el texto.

Añadir el nuevo idioma en Preferences, como se especifica el propio archivo Translate.java.

7. Conclusiones

En este proyecto hemos llevado a cabo la creación de un programa con interfaz gráfica de usuario, de resolución de ecuaciones, aprovechándonos de herramientas libres, para las tareas básicas como el derivado o la evaluación de las ecuaciones.

El código escrito para el proyecto ha superado las 12000 líneas y hemos dedicado unos 8 meses entre la investigación matemática, de librerías, de propiedades termodinámicas y la creación del programa.

El resultado ha sido satisfactorio, ya que cumple todas los objetivos iniciales que nos habíamos planteado:

- Resuelve sistemas de ecuaciones y además lo realiza rápidamente, teniendo en cuenta que en Java las operaciones matemáticas suelen costar, por lo general, el doble de tiempo que en otros lenguajes como C o Fortran[18].
- Permitimos al usuario configurar partes importantes como el punto inicial y la elección del algoritmo matemático.
- Es multiplataforma, está ideado para funcionar hasta en sistemas operativos poco extendidos como BSD o Solaris.
- La documentación, como puede verse en el anexo 5, está completamente en inglés mientras que todo el programa aparece en español e inglés. Además, permite añadir nuevos idiomas.
- Hemos diseñado una interfaz de usuario completa, que permite el guardado de documentos en su propio formato así como imprimir y exportar a PDF.
- Hemos creado una base de datos de propiedades termodinámicas que no representan una lista cerrada, de hecho, el usuario podrá añadir nuevas sustancias y propiedades al programa fácilmente.

Además hemos realizado algunos objetivos no planteados inicialmente, pero que han resultado interesantes a lo largo de todo el proceso, como el programa de cálculo simbólico y el renderizado de ecuaciones.

Los posibles planes de futuro para eSuite serían subir el código y el programa a Internet para permitir que otros accedan a él, ampliar la base de datos, para lo cual no hacen falta conocimientos informáticos, mejorar el programa añadiendo un algoritmo genético que es el tipo de método numérico que usa el Engineering Equation Solver, así como añadirle capacidad para trabajar con tablas, gráficas, un editor de imágenes simple, para poder hacer representaciones gráficas de lo que estamos calculando e incluso una sección de optimización de funciones.

8. Bibliografía

1) **Mohammedi El Hallabi**, *A global convergence theory for arbitrary norm trust region methods for nonlinear equations*, Ph.D, 1987.

2) **William H. Press , Saul A. Teukolsky , William T. Vetterling y Brian P. Flannery,** *Numerical recipes in Fortran 77*, Cambridge University Press, 1986.

3) **Michael W. Trosset**, *Trust Regions and Ridge Analysis*, Department of Mathematics, College of William & Mary.

4) **Oriol Boix**, *Estudio y modelización en r.p. de cargas no lineales para el análisis armónico de redes eléctricas,* 1986.

5) **Ananth Ranganathan,** *The Levenberg-Marquardt Algorithm*, 2004.

6) **P. Rabinowitz,** *Numerical methods for nonlinear algebraic equation*s, Gordon and Breach, 1970.

7) **I.S. Duff y J.K**. **Reid**, *An implementation of Tarjan's algorithm for the block triangularization of a matrix.* 1978.

8) **Robert Tarjan,** *Depth first search and linear graph algorithms.* Siam J. Comp, 1972.

9) **Elmqvist H. y M. Otter**, "Methods for tearing systems of equations in object-oriented modeling" en European Simulation Multiconference. Barcelona, 1994.

10) **Thomas F. Irvine y Peter E. Liley**, *Steam and gas tables with computer equations*, Academic Press, 1984.

11) **K, A. Kobe**, "Termoquímica para la industria petroquímica*"* en *Petroleum Refiner*, 1949-1954.

12) **F. J. Moldes,** *Java SE 6*. Madrid: Anaya, 2007.

13) www.java2s.com

14) www.download.oracle.com

15) www.wikipedia.org

16) **S.A. Klein**, *Development and integration of an equation-solving program for engineering thermodynamic courses*, John Wiley & Sons 1993.

17) www.gnu.org/philosophy/philosophy.es.html

18) **Cherrystone Software Labs**, *Algorithmic Performance Comparison Between C, C++, Java and C# Programming Languages.* 2010.